

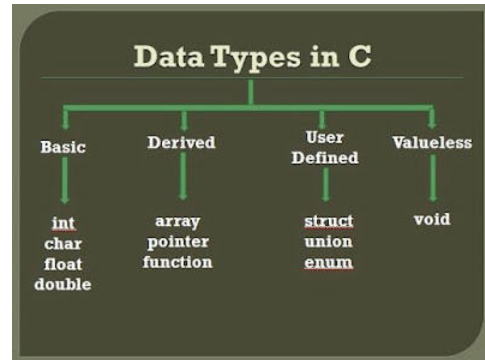
# UNIT – VI

## STRUCTURE, UNION, FILE

### Index

#### **Structure**

- 6.1 Definition and Initialization of Structures,
- 6.2 Accessing Structures,
- 6.3 Nested Structures,
- 6.4 Arrays of Structures,
- 6.5 Structures and Functions,
- 6.6 Pointers to Structures,
- 6.7 Self-Referential Structures,
- 6.8 The Type Definition (**type def**),
- 6.9 **enumerated** types



#### **union**

- 6.10 definition and Initialization of Union,
- 6.11 Accessing of Union.
- 6.12 Differences between structure and union
- 6.13 Programming examples

#### **Files:**

- 6.14.Types of files
- 6.15 File Management
- 6.16 File Modes
- 6.17 Functions to write data to file
- 6.18 Functions to read data from file

# Structures

- ◆ A structure is similar to records.
- ◆ It stores related information about an entity.
- ◆ Structure is basically a user defined data type that can store related information(even of different data types) together.
- ◆ A structure is therefore, a collection of variables under a single name.
- ◆ The variables within a structure are of different data types and each has a name that is used to select it from the structure.

## 1. Structure declaration:

- ◆ A structure is declared using the keyword struct followed by a struct name.
- ◆ All the variables of the structure are declared within the structure.
- ◆ A structure type is generally declared by using the following syntax:

```
struct tag-name
{
    data_type member1;
    data_type member2;
    -----
    -----
};
```

In defining a structure we may note the following syntax:

1. The structure is terminated with a semicolon.
2. While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the structure.
3. The tag name such as employee can be used to declare structure variables of its type, later in the program.

```
struct employee
{
    char name[20];
    char qual[10];
    int empno;
    float salary;
};
```

- ◆ The keyword struct declares a structures to hold the details of four data fields, namely name, qual, empno and salary.
- ◆ These fields are called *structure elements* or *members*.
- ◆ Each member may belong to a different type of data.
- ◆ Employee is the name of the structure and is called the *structure tag*.
- ◆ It simply describes a format called *template* to represent information as shown below:

name	Array of 20 characters
qual	Array of 10 characters
empno	integer
salary	float

## 2. Declaring structure variables:

- ◆ After defining a structure format, we can declare variables of that type.
- ◆ A structure variable declaration is similar to the declaration of variables of any other datatypes.
- ◆ It includes the following elements:
  1. The keyword struct
  2. The structure tag name
  3. List of variable names separated by commas
  4. A terminating semicolon
- ◆ An individual structure type variable can be declared as follows

```
struct tag_name variable1,variable2,.....variableN.
```

- ◆ Struct is a required keyword, tag\_name is the name that appeared in the structure declaration and variable,variable2,..... variableN are structure variables of type tag\_name.
- ◆ We can now declare the structure variables emp1,emp2.

```
struct employee emp1,emp2;
```

Thus emp1 and emp2 are variables of type employee.

Ex 2: It is possible to combine the declaration of the structure with that of structure variables as shown below.

```
struct tag_name  
{  
    member 1;  
    member 2;  
    :  
    member n;  
}var1,var2,.....,varN;
```

Here var1,var2,.....varN are variables. The tag\_name is optional in this situation.

```
struct employee  
{  
    int empno;  
    char empname[10];  
    char qual[10];  
    float sal;  
}emp1,emp2;
```

Thus emp1,emp2 are structure variables of type employee.

The tag employee need not be inclined thus the above declaration can also be written as

```
struct  
{  
    int empno;
```

```
char empname[10];
char qual[10];
float sal;
}emp1,emp2;
```

### 3. ACCESSING THE MEMBERS OF A STRUCTURE:

- ◆ Each member of a structure can be used just like a normal variable.
- ◆ A structure member variable is generally accessed using a '.'(dot) operator.
- ◆ The syntax of accessing a structure or a member of a structure can be given as:

```
struct_var.member_name
```

- ◆ The dot operator is used to select a particular member of the structure. For example, to assign value.

#### Program:

```
#include<stdio.h>
struct emp
{
    int no;
    char name[10];
    float sal;
};
void main()
{
    struct emp e;
    printf("Enter employee number");
    scanf("%d",&e.no);
    printf("Enter employee name");
    scanf("%s",e.name);
    printf("Enter employee salary");
    scanf("%f",&e.sal);
    printf("Employee number=%d\n",e.no);
    printf("Employee name=%s\n",e.name);
    printf("Employee salary=%f\n",e.sal);
}
```

#### Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s4.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter employee number1
Enter employee name Rohith
Enter employee salary 50000
Employee number=1
Employee name=Rohith
Employee salary=50000.000000
```

## INITIALIZATION OF STRUCTURES

- ◆ A structure can be initialized in the same way as other data types are initialized.
- ◆ *Initializing a structure* means assigning some constants to the members of the structure.
- ◆ When the user does not explicitly initialize the structure , then C automatically does that.
- ◆ For int and float members, the values are initialized to zero and char and string members are initialized to the '\0' by default.
- ◆ The initializers are enclosed in braces and are separated by commas.
- ◆ The **general syntax** to initialize a structure variable is given as follows

```
struct struct_name
{
    data_type member_name1;
    data_type member_name2;
    data_type member_name3;
    ...
}struct_var={constant1,constant2,constant3,.....};
or
struct struct_name
struct_var={constant,constant2,constant3,....};
```

**For example**, we can initialize a student structure by writing

```
struct student
{
    int r_no;
    char name[20];
    char course[20];
    float fees;
}stud1={01,"Rohan","BCA",45000};

or by writing

struct student stud1={01, "Rohan", "BCA",45000};
```

## ARRAY OF STRUCTURES

In a class, we do not have just one student, but there may be atleast 30 students. So, the same definition of the structure can be used for all the 30 students. This would be possible when we make an array of the structure. An array of a structure is declared in the same way as we had declared an array of a built in data type.

The general syntax for declaring an array of a structure can be given as,

```
struct struct_name
{
    data_type member_name1;
    data_type member_name2;
    data_type member_name3;
    ...
};
struct struct_name struct_var[index];
```

**Program: To read and display the information of all the students in the class**

```
#include<stdio.h>
struct student
{
    int roll_no;
    char name[80];
    float fees;
    char DOB[80];
};
void main()
{
    struct student stud[50];
    int n,i;
    printf("Enter the no. of students:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the roll number:");
        scanf("%d",&stud[i].roll_no);
        printf("Enter the name:");
        scanf("%s",stud[i].name);
        printf("Enter the fees:");
        scanf("%f",&stud[i].fees);
        printf("Enter the DOB:");
        scanf("%s",stud[i].DOB);
    }
    for(i=0;i<n;i++)
    {
        printf("*****DETAILS OF STUDENT %d*****",i+1);
        printf("\n ROLL No.=%d",stud[i].roll_no);
        printf("\n NAME=%s",stud[i].name);
        printf("\nFEES=%f",stud[i].fees);
        printf("\n DOB=%s",stud[i].DOB);
    }
}
```

### Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s5.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter the no. of students:1
Enter the roll number:1
Enter the name:Rohith
Enter the fees:5678
Enter the DOB:9 9 91
*****DETAILS OF STUDENT 1*****
ROLL No.=1
NAME=Rohith
FEES=5678.000000
DOB=9
```

## NESTED STRUCTURES

A structure can be placed within another structure. i.e, a structure may contain another structure as its member. A structure that contains another structure as its member is called a *nested structure*. Or structure within structure.

The general syntax used for declaration of nested structure is as follows

```
struct tag_name1
{
    datatype member1;
    datatype member2;
    -----
    -----
    -----
};
struct tag_name2
{
    datatype member1;
    -----
    -----
    -----
    struct tag_name1 struct_variable;
};
```

**Program: A program to read and display information of a student, using a structure within a structure**

```
#include<stdio.h>
struct DOB
{
    int day;
    int month;
    int year;
};
struct student
{
    int roll_no;
    char name[100];
    float fees;
    struct DOB date;
};
void main()
{
    struct student stud1;
    printf("Enter the roll number:");
    scanf("%d",&stud1.roll_no);
    printf("Enter the name:");
    scanf("%s",stud1.name);
    printf("Enter the fees:");
    scanf("%f",&stud1.fees);
    printf("Enter the DOB:");
    scanf("%d%d%d",&stud1.date.day,&stud1.date.month,&stud1.date.year);
    printf("\n***STUDENT DETAILS***");
    printf("\nROLL No.=%d",stud1.roll_no);
    printf("\n NAME=%s",stud1.name);
    printf("\n FEES=%f",stud1.fees);
    printf("\n DOB=%d-%d-%d\n",stud1.date.day,stud1.date.month,stud1.date.year);
}
```



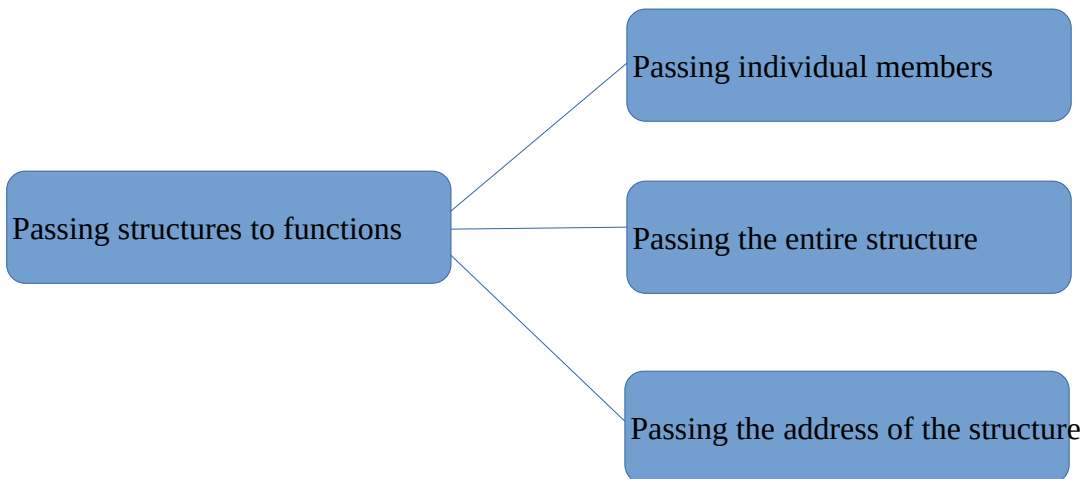
## Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s6.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter the roll number:1
Enter the name:karthikeya
Enter the fees:45000
Enter the DOB:25 09 1991

***STUDENT DETAILS***
ROLL No.=1
NAME=karthikeya
FEES=45000.000000
DOB=25-9-1991
```

# STRUCTURES AND FUNCTIONS

For structures to be fully useful, we must have a mechanism to pass them to functions and return them. A function may access the members of a structure in three ways as shown in figure below.



## 1. Passing individual members:

- ◆ To pass any individual member of the structure to a function, we must use the direct selection operator to refer to the individual members for the actual parameters.
- ◆ The called program does not know if the two variables are ordinary variables or structure members.
- ◆ Look at the following code that illustrates this concept.

```
#include<stdio.h>
struct point
{
    int x;
    int y;
};
void display(int,int);
void main()
{
    struct point p1={2,3};
    display(p1.x,p1.y);
}
void display(int a,int b)
{
    printf("The coordinates of the point are:%d\t%d\n",a,b);
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s7.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
The coordinates of the point are:2      3
```

## 2. PASSING THE ENTIRE STRUCTURE:

- ◆ The second method involves passing of a copy of the entire structure to the called function.
- ◆ Since the function is working on a copy of the structure, any changes to structure members within the function are not reflected in the original structure(in the calling function).
- ◆ It is , therefore, necessary for the function to return the entire structure back to the calling function .
- ◆ All compilers may not support this method of passing the entire structure as a parameter.

The general format of sending a copy of a structure to the called function is:

*function\_name(structure\_variable\_name);*

The called function takes the following form:

```
data_type function_name(struct_type st_name)
{
    .....
    .....
    return(expression);
}
```

**Program :**

```
#include<stdio.h>
struct point
{
    int x;
    int y;
};
void display(struct point);
void main()
{
    struct point p1={2,3};
    display(p1);
}
void display(struct point p)
{
    printf("%d\t%d\n",p.x,p.y);
}
```

**OUTPUT:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s8.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
2      3
```

### 3. PASSING THE ADDRESS OF THE STRUCTURE:

The third approach employs a concept called *pointers* to pass the structure as an argument. In this case, the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it.

The general format of sending address of a structure.

**function\_name(&structure\_variable\_name);**

The called function takes the following form:

```
data_type function_name(struct_type *st_name)
{
    .....
    .....
}
```

**Program:**

```
#include<stdio.h>
struct emp
{
    int no;
    char name[10];
    float sal;
};
void fun(struct emp *p);
void main()
{
    struct emp e;
    printf("Enter employee number:");
    scanf("%d",&e.no);
    printf("Enter employee name:");
    scanf("%s",e.name);
    printf("Enter employee salary:");
    scanf("%f",&e.sal);
    fun(&e);
}
void fun(struct emp *p)
{
    printf("Employee number=%d\n",p->no);
    printf("Employee number=%s\n",p->name);
    printf("Employee salary=%f\n",p->sal);
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s9.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter employee number:111
Enter employee name:Sree
Enter employee salary:25000
Employee number=111
Employee number=Sree
Employee salary=25000.000000
```

## Structures and Pointers

- ◆ Structure can be directly handle by its address by using the pointer with structure.
- ◆ We use pointer with structure through structure variable.
- ◆ The beginning address of the structure can be accessed in the same manner as any other address through the use of address operators(&).
- ◆ The general syntax used for the declaration of pointer with the structure is as

```
struct tag_name
{
    type1 member1;
    type2 member2;
    type3 memberN;
};
struct tag_name v,*p;
```

Where 'v' is the structure variable and 'p' is the pointer variable.

Ex:

```
struct employee
{
    int no;
    char name[10];
    float sal;
};
struct employee emp,*p;
p=&emp;
```

An individual structure member can be accessed in terms of its corresponding pointer variable by writing

**ptvar->member**

where 'ptvar' refer to a structure type pointer variable and the operator(->)

### Program:

```
#include<stdio.h>
struct student
{
    int no;
    char name[10];
    int m;
};
void main()
{
    struct student std;
    struct student *p;
    p=&std;
    printf("enter student number");
    scanf("%d",&p->no);
    printf("enter student name");
    scanf("%s",p->name);
    printf("Enter marks");
    scanf("%d",&p->m);
    printf("Student number=%d\n",p->no);
    printf("Student name=%s\n",p->name);
    printf("marks in m=%d\n",p->m);
}
```

### Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s10.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter student number11
enter student namekrishna
Enter marks507
Student number=11
Student name=krishna
marks in m=507
```

## SELF-REFERENTIAL STRUCTURES

- ◆ Self-referential structures are those structures that contain a reference to data of its same type, i.e, in addition to other data, a self-referential structure contains a pointer to a data that is of the same type as that of the structure.
- ◆ For example, consider the structure node given as follows

```
struct node
{
    int val;
    struct node *next;
};
```

- ◆ Here the structure node will contain two types of data- (1)an integer **val** and (2)**next**, which is a pointer to a node.
- ◆ Actually, self-referential structure is the foundation of other data structures.

## typedef

- ◆ The typedef (derived from type definition) keyword enables the programmer to create a new data type name for an existing datatype.
- ◆ By using typedef, no new data is created, rather an alternate name is given to a known data type.
- ◆ The general syntax of using the **typedef** keyword is given as:

```
typedef existing datatype new data_type;
```

- ◆ Note that the typedef statement doesnot occupy any memory, it simply defines a new type

- ◆ For example, if we write

```
typedef int INTEGER;
```

- ◆ Then INTEGER is the new name of data type int. We may now write

```
INTEGER num=5;
```

- ◆ When we precede a struct name with the typedef keyword, then the struct becomes a new type.

- ◆ A **typedef** declaration is a synonym for the type.

- ◆ For example, writing

```
typedef struct student
{
    int r_no;
    char name[20];
    char course[20];
    float fees;
};
```

- ◆ Now that we have preceded the structure's name with the keyword **typedef**, the student becomes a new data type.

- ◆ Therefore, now we can straightaway declare variables of this new data type as we declare variables of type int, float,char,double,etc., to declare a variable of structure student we will just write,

```
student stud1;
```

- ◆ Note that we have not written struct student stud1.



## Enumeration datatype(enum)

- ◆ An enumeration datatype is a set of values represented by identifiers called enumeration constants.
- ◆ The enumeration constants are specified when the type is defined.
- ◆ The general format of the enumeration data type is,

```
enum user-defined-name
{
    member-1;
    member-2;
    -----
    -----
    member-n;
};
```

- ◆ Where **enum** is a keyword for defining the enumeration data type and the braces are essential.
- ◆ The members of the enumeration data type such as member-1, member-2 and member-n are the individual identifiers.
- ◆ Once the enumeration data type is defined, it can be declared in the following ways:

```
storage-class enum user-defined-name V1,V2,...Vn;
```

- ◆ The enumerated variables V1,V2,...Vn can only have one of the values member-1, member2, .....,member-n.
- ◆ The assignments of the following types are valid:  
V1=member-3;  
V5=member-1;

### Program:

```
#include<stdio.h>
void main()
{
    enum{RED=2,BLUE,BLACK=5, GREEN=7, YELLOW, PURPLE, WHITE=15};
    printf("RED=%d\n",RED);
    printf("BLUE=%d\n",BLUE);
    printf("YELLOW=%d\n",YELLOW);
}
```

### Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc enum.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
RED=2
BLUE=3
YELLOW=8
```

## UNION

- ◆ Unions, like structures, contain members whose individual data types may differ from one another.
- ◆ However the members that, compose a union all share the same storage area within the computer's memory, where as each member within a structure is assigned its own unique storage area.
- ◆ Thus, unions are used to save memory.
- ◆ They are useful for applications involving multiple members, where values need not be assigned to all of the members at any one time.

### 1. DECLARING A UNION:

- ◆ The syntax of declaring a union is the same as that of declaring a structure.
- ◆ The only difference is that instead of using the keyword **struct**, the keyword **union** would be used.
- ◆ The syntax for union declaration can be given as

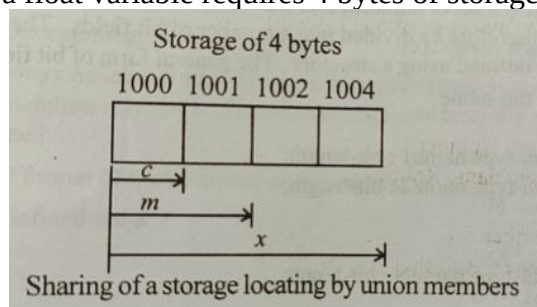
```
union union_name
{
    data_type var-name;
    data_type var-name;
    ...
};
```

Example:

```
union item
{
    int m;
    float x;
    char c;
}code;
```

This declares a variable **code** of type **union item**.

- ◆ The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union.
- ◆ In the declaration above, the **member x** requires 4 bytes which is the largest among the members.
- ◆ Figure shows how all the 3 variables share the same address.
- ◆ This assumes that a float variable requires 4 bytes of storage.



## 2. ACCESSING A MEMBER OF A UNION:

- ◆ A member of a union can be accessed using the same syntax as that of a structure.
- ◆ To access the fields of a union, use the dot operator(.), i.e, the union variable name followed by the dot operator followed by the member name.

**Program:**

```
#include<stdio.h>
union point
{
    int x;
    int y;
};
void main()
{
    union point p;
    p.x=10;
    printf("The x co-ordinate of p is %d\n",p.x);
    p.y=20;
    printf("The y co-ordinate of p is %d\n",p.y);
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc u1.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
The x co-ordinate of p is 10
The y co-ordinate of p is 20
```

# Difference between Structure and Union in C

## structures in C

A structure is a user-defined data type available in C that allows to combining data items of different kinds. Structures are used to represent a record.

**Defining a structure:** To define a structure, we must use the **struct** statement. The struct statement defines a new data type, with more than or equal to one member. The format of the struct statement is as follows:

```
struct [structure name]
{
    member definition;
    member definition;
    ...
    member definition;
};
```

## union

A union is a special data type available in C that allows storing different data types in the same memory location. We can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purposes.

**Defining a Union:** To define a union, we must use the **union** statement in the same way as we did while defining a structure. The union statement defines a new data type with more than one member for our program. The format of the union statement is as follows:

```
union [union name]
{
    member definition;
    member definition;
    ...
    member definition;
};
```

## Similarities between Structure and Union:

1. Both are user-defined data types used to store data of different types as a single unit.
2. Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.
3. Both structures and unions support only assignment = and sizeof operators. The two structures or unions in the assignment must have the same members and member types.
4. A structure or a union can be passed by value to functions and returned by value by functions. The argument must have the same type as the function parameter. A structure or union is passed by value just like a scalar variable as a corresponding parameter.
5. '.' operator is used for accessing members.

## Differences:

	STRUCTURE	UNION
<b>Keyword</b>	The keyword <b>struct</b> is used to define a structure	The keyword <b>union</b> is used to define a union.
<b>Size</b>	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is <b>greater than or equal to the sum of sizes of its members.</b>	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of <b>union is equal to the size of largest member.</b>
<b>Memory</b>	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
<b>Value Altering</b>	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
<b>Accessing members</b>	Individual member can be accessed at a time.	Only one member can be accessed at a time.
<b>Initialization of Members</b>	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

## PROGRAMMING EXAMPLE

### 1. Write a C program to find student grades in a class using structures.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
struct stud
{
    char name[20];
    int marks;
    char grade[3];
};
struct stud s[3];
void main()
{
    int i;
    for(i=1;i<=3;i++)
    {
        printf("Enter %d student name:",i);
        scanf("%s",s[i].name);
        printf("Enter %d student obtained marks=",i);
        scanf("%d",&s[i].marks);
    }
    for(i=1;i<=3;i++)
    {
        if(s[i].marks>=80)
            strcpy(s[i].grade,"A");
        else if(s[i].marks>=60)
            strcpy(s[i].grade,"B");
        else if(s[i].marks>=50)
            strcpy(s[i].grade,"C");
        else if(s[i].marks>=40)
            strcpy(s[i].grade,"D");
        else
            strcpy(s[i].grade,"F");
    }
    for(i=1;i<=3;i++)
        printf("\n%d student %s has obtained grade %s\n",i,s[i].name,s[i].grade);
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc structure.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter 1 student name:Krishna
Enter 1 student obtained marks=65
Enter 2 student name:Rajeev
Enter 2 student obtained marks=80
Enter 3 student name:Ravindra
Enter 3 student obtained marks=55

1 student Krishna has obtained grade B
2 student Rajeev has obtained grade A
3 student Ravindra has obtained grade C
```

## File Handling in C

File handling refers to the method of storing data in the C program in the form of an output or input that might have been generated while running a C program in a data file, i.e., a binary file or a text file for future analysis and reference in that very program.

### **File :**

- ◆ A file refers to a source in which a program stores the information/data in the form of bytes of sequence on a disk (permanently).
- ◆ The content available on a file isn't volatile like the compiler memory in C.
- ◆ But the program can perform various operations, such as creating, opening, reading a file, or even manipulating the data present inside the file.
- ◆ This process is known as file handling in C.

### **Need of File Handling in C:**

There are times when the output generated out of a program after its compilation and running do not serve our intended purpose. In such cases, we might want to check the program's output various times. Now, compiling and running the very same program multiple times becomes a tedious task for any programmer. It is exactly where file handling becomes useful.

Few reasons that file handling makes programming easier for all are:

- **Reusability:** File handling allows us to preserve the information/data generated after we run the program.
- **Saves Time:** Some programs might require a large amount of input from their users. In such cases, file handling allows you to easily access a part of a code using individual commands.
- **Commendable storage capacity:** When storing data in files, you can leave behind the worry of storing all the info in bulk in any program.
- **Portability:** The contents available in any file can be transferred to another one without any data loss in the computer system. This saves a lot of effort and minimises the risk of flawed coding.

## Types of Files in a C Program

When referring to file handling, we refer to files in the form of data files. Now, these data files are available in 2 distinct forms in the C language, namely:

- Text Files
- Binary Files

### **Text Files**

- ◆ The text files are the most basic/simplest types of files that a user can create in a C program.
- ◆ We create the text files using an extension **.txt** with the help of a simple text editor.
- ◆ In general, we can use notepads for the creation of .txt files.
- ◆ These files store info internally in ASCII character format, but when we open these files, the content/text opens in a human-readable form.
- ◆ Text files are, thus, very easy to access as well as use.
- ◆ But there's one major disadvantage; it lacks **security**.
- ◆ Since a .txt file can be accessed easily, information isn't very secure in it.
- ◆ Added to this, text files consume a **very large space** in storage.
- ◆ To solve these problems, we have a different type of file in C programs, known as binary files.

### **Binary Files**

- ◆ The binary files store info and data in the binary format of 0's and 1's (the binary number system).
- ◆ Thus, the files occupy comparatively **lesser space** in the storage.
- ◆ In simpler words, the binary files store data and info the same way a computer holds the info in its memory.
- ◆ Thus, it can be accessed very easily as compared to a text file.
- ◆ The binary files are created with the extension **.bin** in a program, and it overcomes the drawback of the text files in a program since **humans can't read it; only machines can**.
- ◆ Thus, the information becomes much **more secure**.
- ◆ Thus, binary files are safest in terms of storing data files in a C program.



## File management in C

A File can be used to store a large volume of persistent data. Like many other languages 'C' provides following file management functions,

1. Creation of a file
2. Opening a file
3. Reading a file
4. Writing to a file
5. Closing a file

Following are the most important file management functions available in 'C,'

function	purpose
<b>fopen ()</b>	Creating a file or opening an existing file
<b>fclose ()</b>	Closing a file
<b>fprintf ()</b>	Writing a block of data to a file
<b>fscanf ()</b>	Reading a block data from a file
<b>getc ()</b>	Reads a single character from a file
<b>putc ()</b>	Writes a single character to a file
<b>getw ()</b>	Reads an integer from a file
<b>putw ()</b>	Writing an integer to a file
<b>fseek ()</b>	Sets the position of a file pointer to a specified location
<b>ftell ()</b>	Returns the current position of a file pointer
<b>rewind ()</b>	Sets the file pointer at the beginning of a file

### Creating a File

Whenever you want to work with a file, the first step is to create a file. A file is nothing but space in a memory where data is stored.

To create a file in a 'C' program following syntax is used,

```
FILE *fp;  
fp = fopen ("file_name", "mode");
```

In the above syntax, the file is a data structure which is defined in the standard library.

fopen is a standard function which is used to open a file.

- If the file is not present on the system, then it is created and then opened.
- If a file is already present on the system, then it is directly opened using this function.

fp is a file pointer which points to the type file.

Whenever you open or create a file, you have to specify what you are going to do with the file. A file in 'C' programming can be created or opened for reading/writing purposes. A mode is used to specify whether you want to open a file for any of the below-given purposes. Following are the different types of modes in 'C' programming which can be used while working with a file.

## File Mode

## Description

r	Open a file for reading. If a file is in reading mode, then no data is deleted if a file is already present on a system.
w	Open a file for writing. If a file is in writing mode, then a new file is created if a file doesn't exist at all. If a file is already present on a system, then all the data inside the file is truncated, and it is opened for writing purposes.
a	Open a file in append mode. If a file is in append mode, then the file is opened. The content within the file doesn't change.
r+	open for reading and writing from beginning
w+	open for reading and writing, overwriting a file
a+	open for reading and writing, appending to file

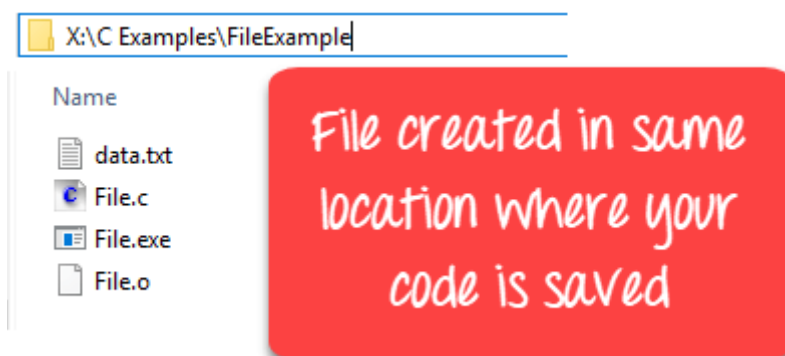
In the given syntax, the filename and the mode are specified as strings hence they must always be enclosed within double quotes.

Example:

```
#include <stdio.h>
int main() {
FILE *fp;
fp = fopen ("data.txt", "w");
}
```

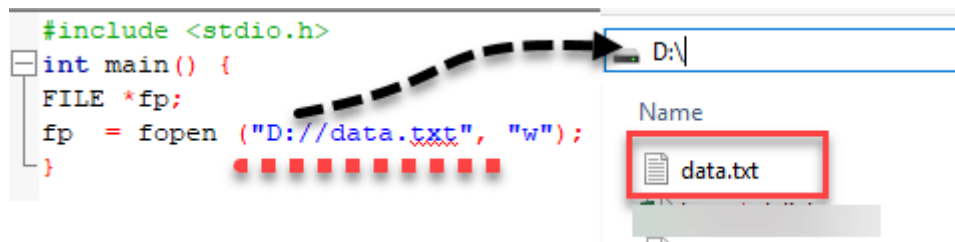
Output:

File is created in the same folder where you have saved your code.



You can specify the path where you want to create your file

```
#include <stdio.h>
int main() {
FILE *fp;
fp = fopen ("D://data.txt", "w");
}
```



## Closing a file

One should always close a file whenever the operations on file are over. It means the contents and links to the file are terminated. This prevents accidental damage to the file.

‘C’ provides the fclose function to perform file closing operation. The syntax of fclose is as follows,

```
fclose (file_pointer);
```

Example:

```
FILE *fp;  
fp = fopen ("data.txt", "r");  
fclose (fp);
```

The fclose function takes a file pointer as an argument. The file associated with the file pointer is then closed with the help of fclose function. It returns 0 if close was successful and EOF (end of file) if there is an error has occurred while file closing.

After closing the file, the same file pointer can also be used with other files.

In ‘C’ programming, files are automatically close when the program is terminated. Closing a file manually by writing fclose function is a good programming practice.

## FILE MODE

- ◆ Mode conveys to C, the type of processing that will be done with the file.
- ◆ The different modes in which a file can be opened for processing are given in table.

Mode	Description
r	Open a text file for reading. If the stream(file) doesnot exist, then an error will be reported.
w	Open a text file for writing. If the stream(file) doesnot exist, then it is created. If the file already exists, then its contents would be deleted.
a	Append to a text file. If the file does not exist, it is created.
r+	Open a text file for both reading and writing. The stream will be positioned at the beginning of the file. The file must already exist.
w+	Open a text file for both reading and writing. The stream will be created if it does not exist, and it will be truncated if it exists.
a+	Open a text file for both reading and appending. The stream will be positioned at the end of the file content.
rb	Open a binary file for reading. B indicates binary. By default this will be a sequential file.
wb	Open a binary file for writing.
ab	Append to a binary file
r+b/rb+	Open a binary file for read/write
w+b/wb+	Create a binary file for read/write
a+b/ab+	Append a binary file for read/append

## Various functions to write data to a file:

C provides the following set of functions to write data to a file:

1. fprintf()-----Formatted output function
2. fputs()-----String output function
3. fputc()-----Character output function
4. fwrite()-----Block write function

### 1. fprintf():

The fprintf() is used to write formatted output to stream. The syntax of the fprintf() can be given as

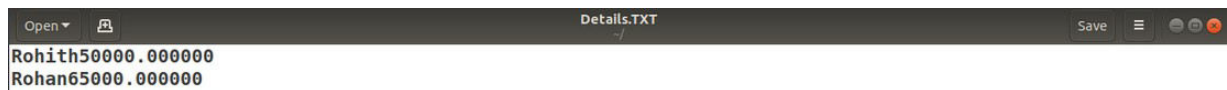
```
int fprintf(FILE *stream,const char *format,...);
```

This function writes data that is formatted as specified by the format argument to the specified stream.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
    FILE *fp;
    int i;
    char name[20];
    float salary;
    fp=fopen("Details.TXT","w");
    if(fp==NULL)
    {
        printf("\n The file could not be opened");
        exit(1);
    }
    for(i=0;i<2;i++)
    {
        puts("Enter your name:");
        scanf("%s",name);
        puts("Enter your salary");
        scanf("%f",&salary);
        fprintf(fp,"%s%f\n",name,salary);
    }
    fclose(fp);
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc files1.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter your name:
Rohith
Enter your salary
50000
Enter your name:
Rohan
Enter your salary
65000
```



Open ▾ Save

Details.TXT

Rohith50000.000000  
Rohan65000.000000

## 2. fputs():

- ◆ The opposite of fgets() is fputs().
- ◆ The fputs() function is used to write a line to a file.
- ◆ The syntax of fputs() can be given as

```
int fputs(const char *str, FILE *stream);
```

- ◆ The fputs() function writes the string pointed to by str to the stream pointed to by stream.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    FILE *fp;
    char feedback[100];
    fp=fopen("Comments.dat","w");
    if(fp==NULL)
    {
        printf("\n The file could not be opened");
        exit(1);
    }
    printf("\nProvide feedback on this book:");
    gets(feedback);
    fputs(feedback,fp);
    fclose(fp);
}
```

### Output:

Provide feedback on this book: good



Open ▾ Save

\*Comments.dat

good

### 3. fputc():

- ◆ The fputc() is used to write a character to the stream.
- ◆ The syntax of fputc() can be given as

```
int fputc(int c, FILE *stream);
```

- ◆ The fputc() function will write the byte specified by c (converted to an unsigned char) to the output stream pointed to by stream.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    FILE *fp;
    char feedback[100];
    int i;
    fp=fopen("comments.dat","w");
    if(fp==NULL)
    {
        printf("\n The file could not be opened");
        exit(1);
    }
    printf("\n Provide feedback on this book:");
    gets(feedback);
    for(i=0;feedback[i]!='\0';i++)
    {
        fputc(feedback[i],fp);
    }
    fclose(fp);
}
```

**Output:**

**Provide feedback on this book: good**



## 4. fwrite():

- ◆ The fwrite() function is used to write data to a file
- ◆ The syntax of fwrite() can be given as

```
int fwrite(const void *str, int size, int count, FILE *stream;
```

- ◆ The fwrite() function will write objects (number of objects will be specified by count) of size, from the array pointed to by ptr to the stream pointed to by stream.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
    FILE *fp;
    int count;
    char str[]="GOOD MORNING";
    fp=fopen("Welcome.txt","wb");
    if(fp==NULL)
    {
        printf("\n The file could not be opened");
        exit(1);
    }
    count=fwrite(str,1,strlen(str),fp);
    printf("\n%d bytes were written to the file",count);
    fclose(fp);
}
```

**Output:**

**13 bytes were written to the file.**





## Various functions to read data from a file:

C provides the following set of functions to read data from a file:

1. fscanf()----- formatted input function
2. fgets()----- String input function
3. fgetc()----- Character input function
4. fread()----- Block read function

### 1. fscanf():

The fscanf() is used to read **formatted data** from the stream. Its syntax can be given as

```
int fscanf(FILE *stream,const char *format,...);
```

The fscanf function is used to read data from the stream and store them according to the parameter format into the locations pointed by the additional arguments.

```
#include<stdio.h>
#include<stdlib.h>
struct student
{
    char name[20];
    float marks;
}stu;
void main()
{
    FILE *fp;
    fp=fopen("students.txt","r");
    if(fp==NULL)
    {
        printf("Error in opening file\n");
        exit(1);
    }
    printf("NAME\tMARKS\n");
    while(fscanf(fp,"%s%f",stu.name,&stu.marks)!=EOF)
        printf("%s\t%f\n",stu.name,stu.marks);
    fclose(fp);
}
```



Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc fscan.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
NAME    MARKS
krishna 48.000000
```

## 2. fgets():

- ◆ The function fgets() stands for *file get string*.
- ◆ This function is used to get a **string** from a stream.
- ◆ The syntax of fgets() can be given as

```
char *fgets(char *str,int size, FILE *stream);
```

- ◆ The fgets() terminates as soon as it encounters either a newline character , EOF, or any other error.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    FILE *fp;
    char str[80];
    fp=fopen("files2.txt","r");
    if(fp==NULL)
    {
        printf("\nThe file could not be opened");
        exit(1);
    }
    while(fgets(str,20,fp)!=NULL)
        printf("\n%s",str);
    printf("\n\nFile Read.Now closing the file");
    fclose(fp);
}
```



### Output:

Yesterday is history  
Tomorrow is mystery  
Today is gift That's  
why it is called Present

File Read.Now closing the file

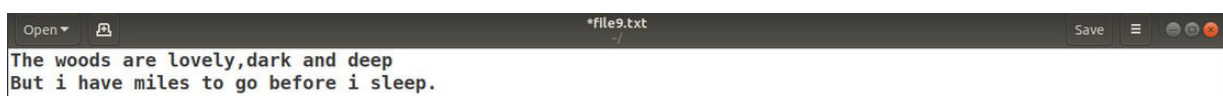
### 3. fgetc():





- ◆ The fgetc() function returns the next **character** from stream, and EOF if the end of file reached, or if there is an error.
- ◆ The syntax of fgetc() can be given as

```
int fgetc(FILE *stream);
```

- ◆ fgetc() reads a single character from the current position of a file(file associated with the stream). After reading the character, the function increments the associated file pointer (if defined) to point the next character.

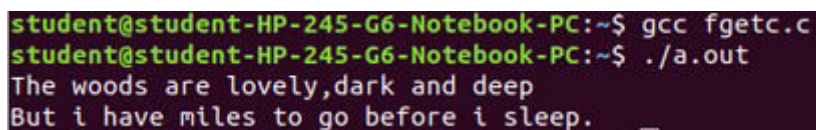
```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    FILE *fp;
    char str[80];
    int i;
    char ch;
    fp=fopen("file9.txt","r");
    if(fp==NULL)
    {
        printf("\n The file could not be opened");
        exit(1);
    }
    ch=fgetc(fp);
    while(ch!=EOF)
    {
        printf("%c",ch);
        ch=fgetc(fp);
    }
    fclose(fp);
}
```

A screenshot of a text editor window titled "file9.txt". The window has a menu bar with "Open", "Save", and other icons. The text content of the file is displayed in a monospaced font: "The woods are lovely,dark and deep" followed by "But i have miles to go before i sleep." on the next line.

Open ▾  \*file9.txt Save   

The woods are lovely,dark and deep  
But i have miles to go before i sleep.

#### Output:

A screenshot of a terminal window. It shows the command to compile the program using gcc, followed by the command to run the resulting executable. The output of the program is displayed, which matches the content of the file9.txt shown in the previous screenshot.

```
student@student-HP-245-G6-Notebook-PC:~$ gcc fgetc.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
The woods are lovely,dark and deep
But i have miles to go before i sleep.
```

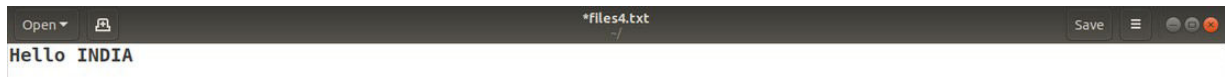
## 4. fread():

- ◆ The fread() function is used to data from a file.
- ◆ Its syntax can be given as,

```
int fread(void *str, int size, int num, FILE *stream);
```

- ◆ The fread() function reads num number of objects(where each object is size bytes) and places them into the array pointed to by str.
- ◆ The data is read from the given input stream.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    FILE *fp;
    char str[11];
    fp=fopen("files4.txt","r+");
    if(fp==NULL)
    {
        printf("The file could not be opened\n");
        exit(1);
    }
    fread(str,1,10,fp);
    str[10]='\0';
    printf("First 9 characters of the file are:%s\n",str);
    fclose(fp);
}
```



### Output:

First 9 characters of the file are: Hello IND